Αλγόριθμοι και Πολυπλοκότητα

1η Σειρά Γραπτών Ασκήσεων

Σεραφείμ Τζελέπης,

AM: el18849

Άσκηση 1)

Η πρώτη άσκηση είναι σε χειρόγραφη μορφή και έχει σκαναριστεί.

Αλγόριθμοι και Πολυπλοκότητα

Σεραφείμ Τζελέπης, Α.Μ.: el18849

1η Σειρά Γραπτών Ασκήσεων

Άσκηση 1) 1) Το n^2 με το $n\log n$ δεν συγκρίνονται με τρόπο που να μπορούμε να χρησιμοποιήσουμε το Master Theorem. Για λόγους απλότητας θεωρούμε ότι το n είναι δύναμη του 2 δηλαδή $n=2^m \Rightarrow \frac{n}{2}=\frac{2^m}{2}=2^{m-1}$

Η αναδρομική σχέση γράφεται λοιπόν ως εξής:

$$T(2^m)=4T(2^{m-1})+\Theta((2^m)^2\log 2^m)$$

$$=4(4T(2^{m-2})+\Theta(4^{m-1}\log 2^{m-1})+\Theta(4^m\log 2^m)$$

$$=4^m T(1)+\Theta\left(4^m\cdot\sum_{K=0}^{K=m}\log 2^K\right)$$

$$=4^m T(1)+\Theta\left(4^m\cdot\frac{1}{2}m\cdot(m+1)\cdot\log 2\right)$$

Και με αντικατάσταση του 2^m έχουμε:

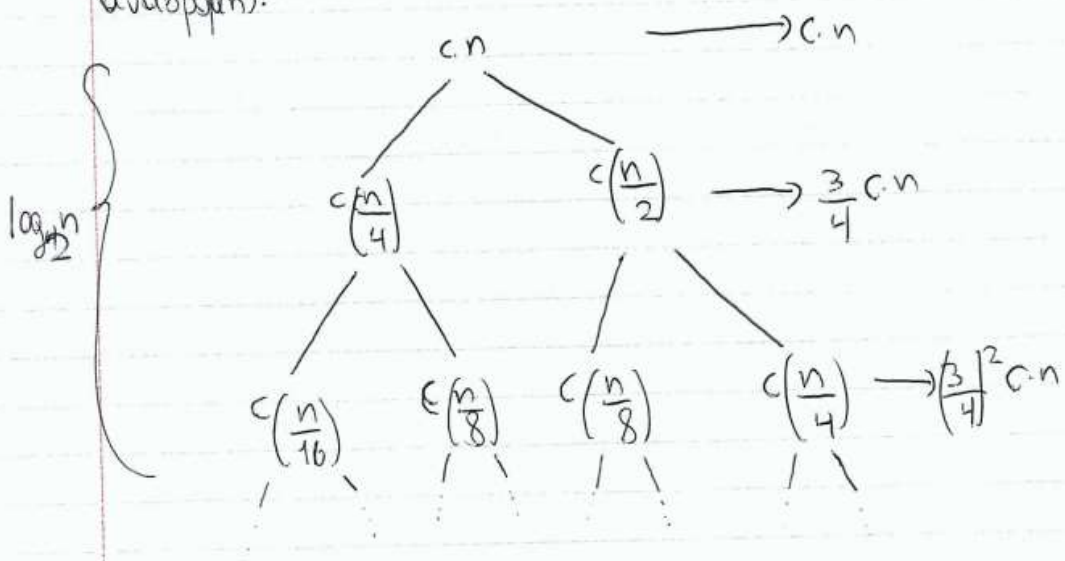
$$T(n)=n^2\cdot T(1)+\Theta\left(n^2\cdot\frac{1}{2}\log n\cdot\log(n+1)\cdot\log 2\right)$$

$$=n^2T(1)+\Theta(n^2\log^2 n)\Rightarrow T(n)=\Theta(n^2\log^2 n)$$

2) Ο όρος που θα χρησιμοποιήσουμε για να ελέγξουμε αν μπορούμε να εφαρμόσουμε το Master Theorem είναι το $n^{\log_b a}=n^{\log_2 5}$ Πράγματι το $n^{\log_2 5}$ είναι μεγαλύτερο από $n^2\log n\,(\Theta(n^2\log n))$ συνεπώς με την χρήση του πρώτου κανόνα προκύπτει ότι:

$$T(n)=\Theta(n^{\log_2 5})$$

3) Η συγκεκριμένη περίπτωση δεν υπόκειται σε Master Theorem. Θα χρησιμοποιήσουμε συνεπώς δέντρο αναδρομής:



$T(1) \cdots T(1) \cdots T(1) \cdots T(1) \cdots T(1) \cdots T(1) \cdots T(1) \cdots T(1) \longrightarrow \Theta(n^{\log_2 2})$

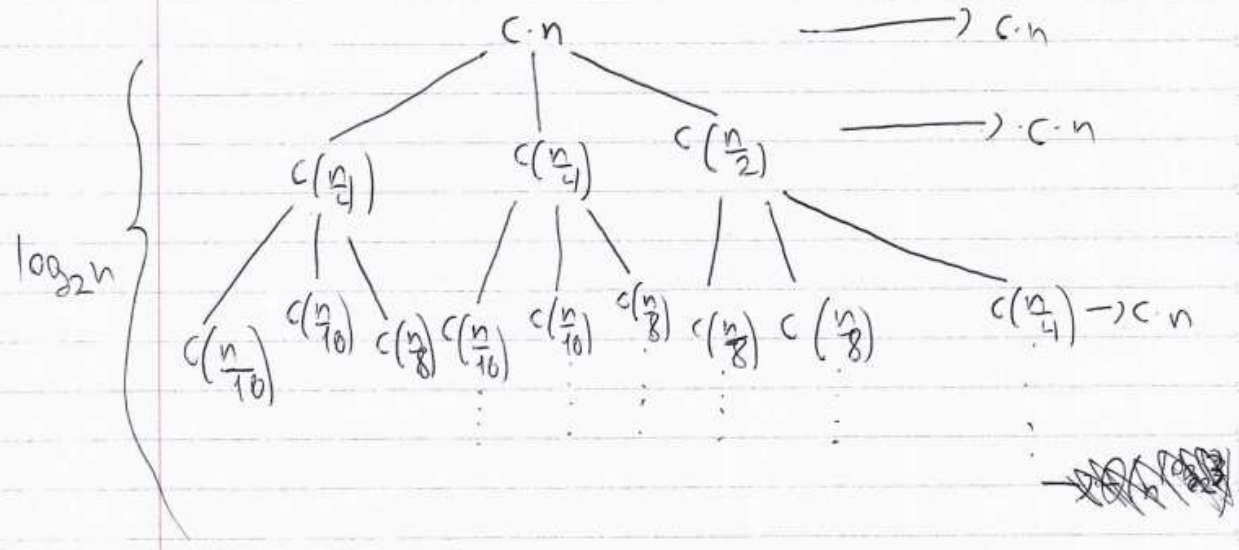
Φτάνουμε στα φύλλα όταν $n/4^i = 1 \Rightarrow i = \log_4 n$ Άρα αυτό είναι το ύψος του δέντρου καθώς $\frac{n}{4} < \frac{n}{2}$, και στο κατώτερο επίπεδο έχουμε $n^{\log_2 2}$ κόμβους.
Το συνολικό κόστος του δέντρου έχει ως εξής:

$$T(n) = cn + \frac{3}{4}cn + \left(\frac{3}{4}\right)^2 \cdot cn + \dots + \Theta(n^{\log_2 2})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{4}\right)^i \cdot cn + \Theta(n^{\log_2 2}) = -4\left(\left(\frac{3}{4}\right)^{\log_4(c)} - 1\right) \cdot cn + \Theta(n)$$

$$= \Theta(n)$$

4) ~~Λάθος~~ Η συγκεκριμένη αναδρομική σχέση δεν υπάκειται σε κάποια περίπτωση του Master Theorem οπότε θα χρησιμοποιήσουμε δέντρο αναδρομής



Βλέπουμε ότι κάθε επίπεδο έχει κόστος $c \cdot n$, συνέπως αφού έχουμε $\log_2 n$ επίπεδα το συνολικό κόστος θα είναι $T(n) = \log_2 n \cdot c \cdot n = \Theta(n \log n)$

5) Ομοίως η αναδρομική σχέση αυτή θα λυθεί με την χρήση του δέντρου αναμονής

$c \cdot \log n \longrightarrow c \cdot \log n$

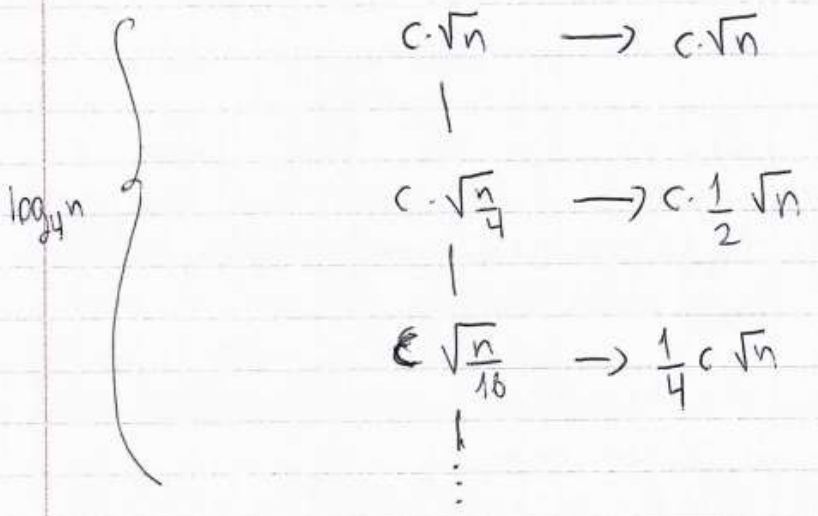
$c \log n\frac{1}{2} \longrightarrow \frac{1}{2} c \cdot \log n$

$c \log n\frac{1}{4} \longrightarrow \left(\frac{1}{2}\right)^2 c \cdot \log n$

5) συνέχεια) Αν το δέντρο της αναδρομικής μας σχέσης είχε άπειρα επίπεδα θα είχε κόστος $\sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i c \log n = 2 \cdot c \cdot \log n = \Theta(\log n)$. Άρα στην περίπτωση μας που το δέντρο θα έχει ως άνω όριο επιπέδων το άπειρο θα ισχύει:

$$T(n) \leq \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i c \log n = \Theta(\log n) \Rightarrow T(n) = \Theta(\log n)$$

6) Για την επίλυση της συγκεκριμένης αναδρομικής σχέσης θα γίνει χρήση του δέντρου ~~αναδρομής~~ αναδρομής



Το συνολικό κόστος θα είναι:

$$T(n) = \sum_{i=0}^{i=\log_4 n} \left(\frac{1}{2}\right)^i c \cdot \sqrt{n} = \Theta(\sqrt{n}) \sum_{i=0}^{i=\log_4 n} \left(\frac{1}{2}\right)^i \leq \Theta(\sqrt{n}) \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

$$\Rightarrow T(n) = \Theta(\sqrt{n})$$

Άσκηση 2)

α)

Ο αλγόριθμος έχει ως εξής:

1. Βρίσκουμε το μεγαλύτερο στοιχείο του Α.
2. Αν το μεγαλύτερο στοιχείο βρίσκεται στην δεξιότερη θέση του Α συνεχίζουμε στο βήμα 5.
3. Αν το μεγαλύτερο στοιχείο βρίσκεται στην αριστερότερη θέση του Α συνεχίζουμε στο βήμα 4, αλλιώς κάνουμε προθεματική περιστροφή ως το μεγαλύτερο στοιχείο, συνεπώς στην αριστερότερη θέση πλέον έχουμε το μεγαλύτερο στοιχείο του Α.
4. Κάνουμε άλλη μια προθεματική περιστροφή φέρνοντας το μεγαλύτερο στοιχείου του Α στην δεξιότερη θέση.
5. Συνεχίζουμε την διαδικασία αναδρομικά(συνολικά θα γίνει n - 1 φόρες) κάθε φόρα στον υποπίνακα Α[0…(n-1)]

Συνεπώς στο τέλος θα έχουμε κάνει το πολύ 2n – 2 προθεματικές περιστροφές και θα έχουμε τον Α ταξινομημένο σε αύξουσα σειρά. Σε κάθε περίπτωση που σε κάποια αναδρομή το μεγαλύτερο στοιχείο βρίσκεται στην αριστερότερη θέση οι συνολικές περιστροφές θα είναι κατά μία λιγότερες από 2n - 2. Ενώ σε κάθε περίπτωση που σε κάποια αναδρομή το μεγαλύτερο στοιχείο βρίσκεται στην δεξιότερη θέση οι συνολικές περιστροφές θα είναι κατά δυο λιγότερες από 2n – 2

Β)

Ο αλγόριθμος για την περίπτωση της προσημασμένης προθεματικής περιστροφής όπου ο Α έχει εξαρχής μόνο θετικά στοιχεία έχει ως εξής:

1. Βρίσκουμε το μεγαλύτερο στοιχείου του Α
2. Αν το μεγαλύτερο στοιχείο του Α βρίσκεται στην δεξιότερη θέση του Α προχωράμε στο βήμα 7.
3. Αν το μεγαλύτερο στοιχείο του Α βρίσκεται στην αριστερότερη θέση κάνουμε μια προσημασμένη προθεματική περιστροφή μήκους 1

ουσιαστικά αντιστρέφοντας απλά το πρόσημο του μέγιστου σε αρνητικό και προχωράμε στο βήμα 5 αλλιώς συνεχίζουμε στο βήμα 4.

4. Φέρνουμε το μεγαλύτερο στοιχείο του Α στην αριστερότερη θέση κάνοντας μια προσημασμένη προθεματική περιστροφή.
5. Φέρνουμε το μεγαλύτερο στοιχείο του Α που πλέον έχει αρνητικό πρόσημο στην δεξιότερη θέση του πίνακα με μια προσημασμένη προθεματική περιστροφή και πλέον στην δεξιότερη θέση βρίσκεται το μεγαλύτερο στοιχείο με θετικό πρόσημο.
6. Εκτελούμε προσημασμένη προθεματική περιστροφή με μήκος την διαφορά του συνολικού μήκους του Α μείον τον αριθμό της θέσης όπου βρήκαμε το μεγαλύτερο στοιχείο αυξημένο κατά 1 (θεωρούμε ότι τα indexes του Α ξεκινάνε από την θέση 0), διορθώνοντας έτσι όλα τα πρόσημα σε θετικά.
7. Συνεχίζουμε την διαδικασία αναδρομικά(συνολικά θα γίνει n – 1 φόρες) κάθε φόρα εξετάζοντας τον πίνακα Α[1…(n-1)].

Έτσι στο τέλος μετά από 3n – 3 προσημασμένες προθεματικές περιστροφές θα έχουμε τον θετικό πίνακα μας ταξινομημένο. Σε κάθε αναδρομή σε περίπτωση που το μεγαλύτερο στοιχείο βρίσκεται στην δεξιότερη θέση γλυτώνουμε 3 περιστροφές.

Γ)

1) Για να δείξουμε ότι μπορούμε να δημιουργήσουμε συμβατά ζεύγη με το πολύ δυο κινήσεις ξεχωρίζουμε δύο περιπτώσεις:

$1^{η}$ Περίπτωση: Να είναι όλοι οι αριθμοί του πίνακα αρνητικοί.

- Σε αυτή την περίπτωση αρχικά γνωρίζουμε από την υπόθεση ότι ο πίνακας δεν θα έχει την μορφή [-1, -2, …, -n] .
- Συνεπώς για κάποιο στοιχείο -x (με x<n) το –(x + 1) θα βρίσκεται αριστερότερα. Αλλιώς ο πίνακας θα είναι ο [-1, -2, …, -n] άρα άτοπο.
- Φέρνουμε το – (x + 1) στην αριστερότερη θέση και πλέον γίνεται (x + 1) και στην συνέχεια με μια ακόμη προσημασμένη προθεματική

περιστροφή μέχρι πριν -x φέρνουμε το (x + 1) μια θέση αριστερότερα από -x και το πρόσημο του πλέον γίνεται αρνητικό όποτε έχουμε το συμβατό ζεύγος (–(x + 1), - x).

2η Περίπτωση: Να υπάρχει θετικός αριθμός στον πίνακα.

- Βρίσκουμε τον μεγαλύτερο θετικό αριθμό x που υπάρχει στον πίνακα.
- Αν x = n, τότε αρκεί με μια περιστροφή να φέρουμε το x στην αριστερότερη θέση και στην συνέχεια με μία ακόμη περιστροφή να το φέρουμε στην δεξιότερη θέση. Το πρόσημο μετά τις δυο περιστροφές θα είναι πάλι θετικό και θα έχουμε το μέγιστο στοιχείο στην δεξιότερη θέση, που σύμφωνα με την εκφώνηση αποτελεί δημιουργία συμβατού ζεύγους και συνεπώς η διαδικασία τελειώνει εδώ αλλιώς αν x ≠ n συνεχίζουμε στο επόμενο βήμα .
- Εφόσον x ≠ n και x ο μεγαλύτερος θετικός αριθμός το x + 1 θα έχει αρνητικό πρόσημο θα είναι δηλαδή –(x + 1).
    - Αν το –(x + 1) βρίσκεται πιο αριστερά από το x, είναι δηλαδή ο πίνακας σε μορφή […, -(x + 1), …, x,…] τότε κάνουμε μια προσημασμένη προθεματική περιστροφή μέχρι και το x και έτσι το φέρνουμε με αρνητικό πρόσημο στην αριστερότερη θέση ενώ επίσης αλλάζει το πρόσημο του (x + 1) και έτσι ο πίνακας πλέον έχει την μορφή [-x, …, (x + 1),…] συνεπώς αρκεί να κάνουμε μια ακόμη προσημασμένη προθεματική περιστροφή μέχρι πριν το (x + 1) και έτσι το x με πλέον θετικό πρόσημο να βρίσκεται μια θέση πριν το (x + 1) δημιουργώντας ένα συμβατό ζεύγος (x, (x + 1)) .
    - Αν το –(x + 1) βρίσκεται πιο δεξιά από το x, είναι δηλαδή ο πίνακας σε μορφή […, x, …, -(x + 1), …] τότε κάνουμε μια προσημασμένη προθεματική περιστροφή μέχρι και το –(x + 1) και έτσι το φέρνουμε με θετικό πρόσημο στην αριστερότερη θέση του πίνακα ενώ πλέον το x έχει αρνητικό πρόσημο και βρίσκεται στα δεξιά του, ο πίνακας γίνεται λοιπόν [(x + 1),…,-x, ...]. Αρκεί συνεπώς να κάνουμε ακόμη μια προσημασμένη

προθεματική περιστροφή μέχρι πριν το -x και έτσι το (x + 1) θα βρίσκεται πλέον μια θέση αριστερά από -x και με αρνητικό πρόσημο, […, -(x + 1), -x, …] , και άρα έχουμε δημιουργήσει ένα συμβατό ζεύγος (-(x + 1), -x).

2)

Ο αλγόριθμος με τον όποιο μπορούμε να ταξινομήσουμε τον A έχει ως εξής:

1. Απαλείφουμε όλα τα συμβατά ζεύγη αναδρομικά, αντικαθιστώντας τα με ένα προσημασμένο στοιχείο. Συνεπώς προκύπτει νέο πρόβλημα το όποιο δεν έχει συμβατά ζεύγη και έχει μικρότερο πλήθος μη απαλειμμένων στοιχείων y. Όταν αυτό το πλήθος y είναι 0 σημαίνει ότι όλα τα ζεύγη έχουν δημιουργηθεί αναδρομικά έχοντας έτσι τον πίνακα μας ταξινομημένο.
2. Αν ο πίνακας είναι ο [-1, -2, …, -y], τότε για y φόρες κάνουμε τα εξής δυο βήματα:
    - Εκτελούμε μια προσημασμένη προθεματική περιστροφή όλων των στοιχείων.
    - Εκτελούμε μια προσημασμένη προθεματική περιστροφή των αριστερότερων y – 1 στοιχείων

    Μετά την εκτέλεση των δυο βημάτων y φορές ο πίνακας μας έχει ταξινομηθεί.
3. Αλλιώς δημιουργούμε νέο συμβατό ζεύγος με την διαδικασία που εξηγήθηκε στο ερώτημα 1 και αφού το κατασκευάσουμε πάμε στο βήμα 1 του αλγόριθμου.

Για την ολοκλήρωση της ταξινόμησης αρκεί να έχουμε δημιουργήσει n συμβατά ζεύγη. Αν k το πλήθος των επαναλήψεων του αλγόριθμου, τότε οι κινήσεις μας θα είναι $2k + 2(n – k) = 2n$ διότι σε κάθε επανάληψη κάνουμε δυο κινήσεις και κάθε μια από αυτές μειώνει το πλήθος των ζευγών τουλάχιστον κατά 1

Άσκηση 3)

Για να μπορέσουμε να λύσουμε αυτό το πρόβλημα με αποδοτικό τρόπο θα χρησιμοποιήσουμε την δομή της στοίβας. Αρχικά θα έχουμε σε έναν πίνακα τα δεδομένα μας, έναν νέο πίνακα αρχικοποιήμενο στο -1 για κάθε στοιχείο με μέγεθος όσο και ο πίνακας με τα δεδομένα μας και επίσης θα έχουμε μια στοίβα, αρχικώς κενή προφανώς. Στην στοίβα θα αποθηκεύουμε, τα indexes των στοιχείων και όχι την τιμή του στοιχείου, όποτε στις συγκρίσεις που θα γίνονται μετά εννοείτε ότι η σύγκριση δεν γίνεται με το ίδιο το στοιχείο της στοίβας, ένα index δηλαδή, αλλά με το στοιχείο του πίνακα δεδομένων μας που αντιστοιχεί στην θέση του index. Στην συνέχεια θα ελέγχουμε ένα ένα τα στοιχεία μας του πίνακα δεδομένων μας ξεκινώντας από το τέλος προς την αρχή(από τα αριστερά προς τα δεξιά). Για κάθε στοιχείο που ελέγχουμε(ή αλλιώς πραγματευόμαστε καθώς διατρέχουμε ανάποδα τον πίνακα μας) θα κάνουμε pop όλα τα στοιχεία από την κορυφή της στοίβας τα οποία είναι μικρότερα, και για το index το όποιο αντιστοιχεί στα στοιχεία της στοίβας στο ίδιο index αυτό θα πηγαίνουμε στον νέο πίνακα όπου αποθηκεύουμε τις κυρίαρχες θέσεις κάθε στοιχείου και θα βάζουμε σαν τιμή το index του στοιχείο του πίνακα το όποιο ελέγχαμε και ήταν μεγαλύτερο από αυτά της στοίβας, αφού γίνουν pop από την κορυφή της στοίβας όσα στοιχεία ήταν μικρότερα από το current, τότε κάνουμε push το index του στοιχείου που πραγματευόμασταν μέχρι τώρα. Έτσι κάθε στοιχείο του πίνακα το βάζουμε στην στοίβα από την οποία θα βγει μόνο όταν βρεθεί στοιχείο μεγαλύτερο από αυτό, και επειδή πηγαίνουμε από δεξιά προς τα αριστερά το στοιχείο το όποιο θα οδηγήσει στο pop θα είναι το πιο «κοντινό» από τα αριστερά το οποίο είναι μεγαλύτερο και του συνεπώς η θέση του αποτελεί την ζητούμενη κυρίαρχη θέση η οποία, όπως προαναφέρθηκε αποθηκεύετε στον νέο πίνακα. Εννοείτε ότι η θέση 0 του νέου πίνακα θα είναι -1, δηλαδή η αρχικοποιημένη τιμή καθώς δεν ορίζεται κυρίαρχη θέση για το πρώτο στοιχείο, εφόσον δεν υπάρχει κάποιο στοιχείο στα αριστερά του. Ενώ στις άλλες θέσεις του νέου πίνακα ,αφού ολοκληρωθεί ο αλγόριθμος μας, θα υπάρχουν σαν τιμές οι κυρίαρχες θέσεις για κάθε στοιχείο. Συνεπώς αφού διατρέχουμε μια φόρα τον πίνακα και από τη στοίβα θα κάνουμε pop όσες φόρες όσα και το πλήθος των στοιχείων του πίνακα, η πολυπλοκότητα του αλγορίθμου αυτού είναι O(n).

Άσκηση 4)

Αρχικά στο συγκεκριμένο πρόβλημα έχουμε δεχθεί κάποιες παραδοχές:

- Η συνολική καθυστέρηση ενός αυτοκινήτου περιλαμβάνει το χρονικό διάστημα από τη στιγμή την όποια συνέβη η άφιξη μέχρι και τη χρονική στιγμή για την οποία ολοκληρώθηκε η εξυπηρέτηση, δηλαδή το συνολικό χρονικό διάστημα είναι ο χρόνος αναμονής + 1.
- Οι αφίξεις γίνονται σε ακέραιες χρονικές στιγμές μόνο.
- Η μέγιστη αποδεκτή καθυστέρηση d είναι δεδομένη και αποτελεί τον χρόνο που προαναφέρθηκε παραπάνω.

Ξεκινώντας λοιπόν, υλοποιούμε έναν νέο πίνακα, μήκους T και αρχικοποιημένος με μηδενικά, ο όποιος σε κάθε θέση του θα έχει το πλήθος των στοιχείων που αφίχθηκαν την χρονική στιγμή που αντιστοιχεί στην θέση αυτή. Πιο συγκεκριμένα το στοιχείο του πίνακα αυτού, έστω ότι τον ονομάζουμε A, A[i] έχει ως τιμή τον αριθμό των αυτοκινήτων τα οποία έφθασαν την χρονική στιγμή i. Συνεπώς η δημιουργία αυτού του πίνακα γίνεται εύκολα διατρέχοντας τον πίνακα με τις αφίξεις αυξάνοντας κάθε φόρα κατά ένα το στοιχείου του νέου πίνακα που έχει ως index την τιμή του πίνακα με τις αφίξεις. Η διαδικασία αυτή έχει πολυπλοκότητα O(n) και έχει ως αποτέλεσμα το πίνακα με τον αριθμό των αφίξεων ανά χρονική στιγμή, μήκους T. Στην συνέχεια πριν φθάσουμε στην εκτέλεση του αλγόριθμου θα υλοποιήσουμε μια νέα συνάρτηση η οποία θα έχει ως σκοπό, για δεδομένο πίνακα με το σύνολο αφίξεων ανά χρονική στιγμή και δεδομένο αριθμό υποδοχέων να ελέγχει αν ο μέγιστος χρόνος καθυστέρησης με τους συγκεκριμένους υποδοχείς θα υπερβεί τον χρόνο d. Η συνάρτηση αυτή δουλεύει ως εξής, διατρέχει τον πίνακα με το σύνολο των αφίξεων και κοιτάει ένα τα στοιχεία του από τα αριστερά προς τα δεξιά, κάθε φόρα που ένα στοιχείο έχει τιμή μεγαλύτερη από αυτή των υποδοχέων, δηλαδή έχουμε περισσότερες αφίξεις από υποδοχείς τότε υπολογίζουμε την μέγιστη καθυστέρηση των αμαξιών που περισσεύουν. Αυτή η καθυστέρηση υπολογίζεται ως το πλήθος των περισσευούμενων δια τους υποδοχείς(καθυστέρηση αναμονής) + 1 (χρόνος εξυπηρέτησης). Αν αυτή η καθυστέρηση υπερβαίνει το d τότε δεν έχουμε επαρκείς υποδοχείς, αλλιώς προσθέτουμε τα περισσευούμενα αυτοκίνητα στην επόμενη ως αφίξεις στην επόμενη χρονική στιγμή του πίνακα μας και συνεχίζουμε να διατρέχουμε τον πίνακα, και στο τέλος αν έχοντας διατρέξει το

πίνακα δεν έχουμε βρει μη αποδεκτή καθυστέρηση η συνάρτηση επιστρέφει ότι οι υποδοχείς είναι επαρκείς. Αυτή η βοηθητική συνάρτηση έχει πολυπλοκότητα O(T) καθώς διατρέχουμε τον πίνακα συνόλου αφίξεων ανά χρονική στιγμή που έχει μέγεθος T. Τώρα έχοντας έτοιμο τον πίνακα συνόλου αφίξεων και την βοηθητική συνάρτηση ελέγχου μπορούμε να δούμε τον αλγόριθμο. Ουσιαστικά θα κάνουμε δυαδική αναζήτηση για τον αριθμό υποδοχέων, με κάτω άκρο το 0 και άνω άκρο το n(σίγουρα αρκούν οι n υποδοχείς καθώς έτσι η έτσι αντιστοιχεί ένας υποδοχέας ανά αμάξι). Άρα ξεκινάμε καλώντας την συνάρτηση ελέγχου για αριθμό υποδοχέων n/2, αν επαρκούν αυτοί οι υποδοχείς τότε κάνουμε δυαδική αναζήτηση στο διάστημα 0,n/2 αλλιώς αν χρειαζόμαστε περισσότερους κοιτάμε στο διάστημα n/2,n. Εκτελούμε συνεπώς δυαδική αναζήτηση και κάθε φόρα που επιτυγχάνουμε κρατάμε την τελευταία τιμή για την οποία είχαμε επιτυχία. Η δυαδική αναζήτηση συνεχίζεται μέχρι να φτάσουμε σε ένα μοναδιαίο διάστημα, αν η μικρότερη τιμή αυτού του μοναδιαίου διαστήματος είναι αυτή της τελευταίας επιτυχίας που κρατάμε τότε βρήκαμε τον ελάχιστο αριθμό, αλλιώς αν η τελευταία επιτυχία είναι η τιμή της μεγαλύτερης τιμής του μοναδιαίου διαστήματος αυτού θα ελέγξουμε αν η μικρότερη τιμή, είναι επαρκής αριθμός υποδοχέων και αν είναι θα ανανεώσουμε την τιμή, όπως και να έχει μετά και αυτόν τον έλεγχο έχουμε βρει τον ελάχιστο αριθμό υποδοχέων. Συνολικά δηλαδή έχουμε τον κορμό μας ο οποίος εκτελεί δυαδική αναζήτηση πολυπλοκότητας logn και για κάθε τιμή ελέγχει αν είναι επαρκής συνεπώς εκτελούμε κάθε φόρα έναν έλεγχο πολυπλοκότητας O(T). Συνεπώς η τελική πολυπλοκότητα του αλγόριθμου μας είναι O(T*logn).

Άσκηση 5)

Α) Για το πρώτο ερώτημα θα χρησιμοποιήσουμε την μέθοδο της δυαδικής αναζήτησης. Το αρχικό διάστημα στο οποίο θα την εφαρμόσουμε είναι το 0 έως M καθώς γνωρίζουμε ότι το μέγιστο στοιχείο του πολυσυνόλου S είναι μικρότερο ή ίσο του M. Συνεπώς ο αλγόριθμος μας έχει ως εξής. Αρχικά ξεκινάμε και εφαρμόζουμε στο ενδιάμεσο του διαστήματος μας M/2 την Fs. Αν Fs(M/2) >= k τότε το διάστημα στο οποίο θα ψάξουμε, σύμφωνα με την δυαδική αναζήτηση, θα είναι το 0 έως M/2, αλλιώς αν Fs(M/2) < k τότε θα συνεχίσουμε στο διάστημα

M/2 έως M. Δηλαδή εκτελούμε δυαδική αναζήτηση και κάθε φόρα που η Fs του μέσο του διαστήματος είναι μεγαλύτερη ή ίση του πηγαίνουμε στο χαμηλότερο μισό του διαστήματος ενώ αν η Fs είναι μικρότερη πηγαίνουμε στο μεγαλύτερο μισό του διαστήματος. Κάθε φόρα που η Fs επιστρέφει αποτέλεσμα μεγαλύτερο ή ίσο του k ανανεώνουμε μία μεταβλητή με την τιμή για την οποία κλήθηκε η Fs. Συνεπώς αυτή η μεταβλητή έστω y θα έχει την μικρότερη τιμή για την οποία Fs(y) >= k. Η δυαδική αναζήτηση θα τερματίσει όταν καταλήξουμε σε μοναδιαίο διάστημα, αν η μικρότερη τιμή αυτού του μοναδιαίου διαστήματος είναι αυτή της τελευταίας επιτυχίας(Fs(y)>=k) που κρατάμε τότε βρήκαμε τον ελάχιστο αριθμό και είναι ο y, αλλιώς αν y είναι το άνω άκρο του μοναδιαίου διαστήματος τότε θα πρέπει να ελέγξουμε και το κάτω άκρο. Αν για το κάτω άκρο ισχύει Fs >= k τότε ανανεώνουμε το y και συνεπώς το k-οστό μικρότερο ψηφίο είναι το ανανεωμένο y, αλλιώς αν Fs(κάτω άκρου) < k τότε δεν ανανεώνουμε το y, και επομένως το k-οστό μικρότερο ψηφίο είναι το μη-ανανεωμένο y. Η αναζήτηση δεν σταματάει όταν για κάποιο x ισχύει ότι Fs(x) = k διότι αυτό το x μπορεί να μην ανήκει στο S και συνεπώς να υπάρχει κάποιο l , l < x για το όποιο επίσης να ισχύει Fs(l) = k. Επίσης υπάρχει και η περίπτωση που για το k-οστό μικρότερο στοιχείο y να μην ισχύει Fs(y) = k καθώς μπορεί να υπάρχουν πολλά y στο πολυσύνολο. Οπότε καταλαβαίνουμε ότι αλγόριθμος μας είναι σωστός και λαμβάνει όλες αυτές τις περιπτώσεις και η πολυπλοκότητα του είναι log(M) ,δηλαδή θα κάνουμε logM κλήσεις της Fs ,καθώς εκτελούμε μια εξαντλητική δυαδική αναζήτηση σε διάστημα μήκους M.

B)

Αποδείξαμε στο παραπάνω ερώτημα ότι για τον υπολογισμό του k-οστού μικρότερου στοιχείου θα γίνουν logM κλήσεις της Fs. Επομένως ο συνολικός αλγόριθμος θα έχει πολυπλοκότητα το γινόμενο της πολυπλοκότητας της Fs με logM. Αρχικά θα ταξινομήσουμε τον πίνακα A μήκους n συνεπώς η ταξινόμηση αυτή θα έχει πολυπλοκότητα O(nlogn). Αφού ταξινομήσουμε τον πίνακα ο

αλγόριθμος για την Fs,έστω ότι έχει κληθεί για έναν αριθμό λ, έχει ως εξής, θεωρούμε δύο δείκτες έναν i και έναν j. Με τον δείκτη i θα διατρέξουμε τον πίνακα Α και κάθε φόρα θα «προχωράμε» τον δείκτη j κατά μια θέση δεξιά μέχρι η διαφορά A[j] - A[i] > λ όταν για ένα συγκεκριμένο i βρίσκουμε αυτό το ελάχιστο j για το οποίο η διαφορά τους είναι θετική τότε προσθέτω σε μια μεταβλητή ,στην οποία μετράμε το πλήθος των διαφορών μικρότερες από λ, τον αριθμό j – i – 1 που είναι ουσιαστικά το πλήθος των διαφορών που ξεκινάνε από το i και είναι μικρότερες οι ίσες του λ. Έτσι στο τέλος αφού διατρέξουμε τον πίνακα Α με θα έχουμε το πλήθος των διαφορών που είναι μικρότερες από k ξεκινώντας από όλα τα στοιχεία αφού το έχουμε κάνει για όλα τα i. Όμως επειδή ο πίνακας είναι ταξινομημένος το j θα διατρέξει μόνο μια φόρα τον πίνακα απο αριστερά προς τα δεξιά για όλα τα i, συνεπώς θα διατρέξουμε μια φορά τον πίνακα με το i και μια με το j άρα συνολικά θα έχουμε πολυπλοκότητα O(2n) για την Fs. Επομένως η συνολική πολυπλοκότητα όπως προαναφέρθηκε παραπάνω είναι O(2nlogM) δεδομένου ότι ο πίνακας Α είναι ταξινομημένος αλλιώς θα έχουμε O(2nlogM) + O(nlogn).